20 iOS Developer Interview Questions

Getting ready for your iOS interview

The big day is coming up. Whether it's a phone, online or in person interview it is always a little bit stressful. So, to ease the stress you did some prep work- completed the homework on the company you are interviewing with, stalked the hiring manager and half of the engineering team at the company of your choice on linkedin & github, you brushed up on some of the skills and knowledge you haven't used in a while- you are ready to go.

But following US Navy Seals motto **"Two is one and one is none"** let's go through another quick refresher to help you ace that iOS interview. So let's get right into it!

Let's start with some of the basic questions that I hope you are already ready for. They might seem obvious, but you would be shocked how many engineers failed miserably at them during the 12 years I have been doing technical interviews.

Technical Questions

Ok, now that we've got that covered, let's jump into the technical questions.

Question 1

On a UITableViewCell constructor:

```
- (id)initWithStyle:(UITableViewCellStyle)style reuseIdentifier:(NSString
*)reuseIdentifier
```

What is the `reuseIdentifier` used for?

The `reuseIdentifier` is used to indicate that a cell can be re-used in a `UITableView`. For example when the cell looks the same, but has different content. The `UITableView` will maintain an internal cache of `UITableViewCell`'s with the `reuseIdentifier` and allow them to be re-used when `dequeueReusableCellWithIdentifier:` is called. By re-using table cell's the scroll performance of the tableview is better because new views do not need to be created.

Question 2

Explain the difference between atomic and nonatomic synthesized properties?

Atomic and non-atomic refers to whether the setters/getters for a property will atomically read and write values to the property. When the atomic keyword is used on a property, any access to it will be "synchronized". Therefore a call to the getter will be guaranteed to return a valid value, however this does come with a small performance penalty. Hence in some situations nonatomic is used to provide faster access to a property, but there is a chance of a race condition causing the property to be nil under rare circumstances (when a value is being set from another thread and the old value was released from memory but the new value hasn't yet been fully assigned to the location in memory for the property).

Question 3

Explain the difference between copy and retain?

Retaining an object means the retain count increases by one. This means the instance of the object will be kept in memory until it’s retain count drops to zero. The property will store a reference to this instance and will share the same instance with anyone else who retained it too. Copy means the object will be cloned with duplicate values. It is not shared with any one else.

Question 4

What is method swizzling in Objective C and why would you use it?

Method swizzling allows the implementation of an existing selector to be switched at runtime for a different implementation in a classes dispatch table. Swizzling allows you to write code that can be executed before and/or after the original method. For example perhaps to track the time method execution took, or to insert log statements

```
#import "UIViewController+Log.h"
@implementation UIViewController (Log)
    + (void)load {
        static dispatch_once_t once_token;
        dispatch_once(&once_token,  ^{
            SEL viewWillAppearSelector = @selector(viewDidAppear:);
            SEL viewWillAppearLoggerSelector = @selector(log_viewDidAppear:);
            Method originalMethod = class_getInstanceMethod(self,
viewWillAppearSelector);
            Method extendedMethod = class_getInstanceMethod(self,
viewWillAppearLoggerSelector);
            method_exchangeImplementations(originalMethod, extendedMethod);
        });
    }
    - (void) log_viewDidAppear:(BOOL)animated {
        [self log_viewDidAppear:animated];
        NSLog(@"viewDidAppear executed for %@", [self class]);
    }
@end
```

Question 5

What’s the difference between not-running, inactive, active, background and suspended execution states?

- **Not running:** The app has not been launched or was running but was terminated by the system.
- **Inactive:** The app is running in the foreground but is currently not receiving events. (It may be executing other code though.) An app usually stays in this state only briefly as it transitions to a different state.
- **Active:** The app is running in the foreground and is receiving events. This is the normal mode for foreground apps.
- **Background:** The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended. However, an app that requests extra execution time may remain in this state for a period of time. In addition, an app being launched directly into the background enters this state instead of the inactive state.

- **Suspended:** The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute any code. When a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app.

Question 6

What is a category and when is it used?

A category is a way of adding additional methods to a class without extending it. It is often used to add a collection of related methods. A common use case is to add additional methods to built in classes in the Cocoa frameworks. For example adding async download methods to the `UIImage` class.

Question 7

Can you spot the bug in the following code and suggest how to fix it:

```
@interface MyCustomController : UIViewController

@property (strong, nonatomic) UILabel *alert;

@end

@implementation MyCustomController

- (void)viewDidLoad {
     CGRect frame = CGRectMake(100, 100, 100, 50);
     self.alert = [[UILabel alloc] initWithFrame:frame];
     self.alert.text = @"Please wait...";
     [self.view addSubview:self.alert];
      dispatch_async(
        dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0),
        ^{
           sleep(10);
           self.alert.text = @"Waiting over";
        }
    );
}

@end
```

All UI updates must be done on the main thread. In the code above the update to the alert text may or may not happen on the main thread, since the global dispatch queue makes no guarantees . Therefore the code should be modified to always run the UI update on the main thread

```
dispatch_async(
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0),
    ^{
      sleep(10);
      dispatch_async(dispatch_get_main_queue(), ^{
         self.alert.text = @"Waiting over";
    });
});
```

Question 8

What is the difference between `viewDidLoad` and `viewDidAppear`? Which should you use to load data from a remote server to display in the view?

`viewDidLoad` is called when the view is loaded, whether from a Xib file, storyboard or programmatically created in `loadView`. `viewDidAppear` is called every time the view is presented on the device. Which to use depends on the use case for your data. If the data is fairly static and not likely to change then it can be loaded in `viewDidLoad` and cached. However if the data changes regularly then using `viewDidAppear` to load it is better. In both situations, the data should be loaded asynchronously on a background thread to avoid blocking the UI.

Question 9

What considerations do you need when writing a `UITableViewController` which shows images downloaded from a remote server?

This is a very common task in iOS and a good answer here can cover a whole host of knowledge. The important piece of information in the question is that the images are hosted remotely and they may take time to download, therefore when it asks for "considerations", you should be talking about:

- Only download the image when the cell is scrolled into view, i.e. when `cellForRowAtIndexPath` is called.
- Downloading the image asynchronously on a background thread so as not to block the UI so the user can keep scrolling.
- When the image has downloaded for a cell we need to check if that cell is still in the view or whether it has been re-used by another piece of data. If it's been re-used then we should discard the image, otherwise we need to switch back to the main thread to change the image on the cell.

Other good answers will go on to talk about offline caching of the images, using placeholder images while the images are being downloaded.

Question 10

What is a protocol, how do you define your own and when is it used?

A protocol is similar to an interface from Java. It defines a list of required and optional methods that a class must/can implement if it adopts the protocol. Any class can implement a protocol and other classes can then send messages to that class based on the protocol methods without it knowing the type of the class.

```
@protocol MyCustomDataSource
- (NSUInteger)numberOfRecords;
- (NSDictionary *)recordAtIndex:(NSUInteger)index;
@optional
- (NSString *)titleForRecordAtIndex:(NSUInteger)index;
@end
```

A common use case is providing a DataSource for `UITableView` or `UICollectionView`.

Question 11

What is KVC and KVO? Give an example of using KVC to set a value.

KVC stands for *Key-Value Coding*. It's a mechanism by which an object's properties can be accessed using string's at runtime rather than having to statically know the property names at development time. *KVO* stands for *Key-Value Observing* and allows a controller or class to observe changes to a property value.

Let's say there is a property `name` on a class:

```
@property (nonatomic, copy) NSString *name;
```

We can access it using KVC:

```
NSString *n = [object valueForKey:@"name"]
```

And we can modify it's value by sending it the message:

```
[object setValue:@"Mary" forKey:@"name"]
```

Question 12

What are blocks and how are they used?

Blocks are a way of defining a single task or unit of behavior without having to write an entire Objective-C class. Under the covers Blocks are still Objective C objects. They are a language level feature that allow programming techniques like lambdas and closures to be supported in Objective-C. Creating a block is done using the `^ { }` syntax:

```
 myBlock = ^{
    NSLog(@"This is a block");
 }
```

It can be invoked like so:

```
myBlock();
```

It is essentially a function pointer which also has a signature that can be used to enforce type safety at compile and runtime. For example you can pass a block with a specific signature to a method like so:

```
- (void)callMyBlock:(void (^)(void))callbackBlock;
```

If you wanted the block to be given some data you can change the signature to include them:

```
- (void)callMyBlock:(void (^)(double, double))block {
    ...
    block(3.0, 2.0);
}
```

Question 13

What mechanisms does iOS provide to support multi-threading?

- `NSThread` creates a new low-level thread which can be started by calling the `start` method.

```
NSThread* myThread = [[NSThread alloc] initWithTarget:self
```

```
                                            selector:@selector(myThreadMainMethod:)
                                            object:nil];
[myThread start];
```

- `NSOperationQueue` allows a pool of threads to be created and used to execute `NSOperation`s in parallel. `NSOperation`s can also be run on the main thread by asking `NSOperationQueue` for the `mainQueue`.

```
NSOperationQueue* myQueue = [[NSOperationQueue alloc] init];
[myQueue addOperation:anOperation];
[myQueue addOperationWithBlock:^{

}];
```

- *GCD* or *Grand Central Dispatch* is a modern feature of Objective-C that provides a rich set of methods and API's to use in order to support common multi-threading tasks. *GCD* provides a way to queue tasks for dispatch on either the main thread, a concurrent queue (tasks are run in parallel) or a serial queue (tasks are run in FIFO order).

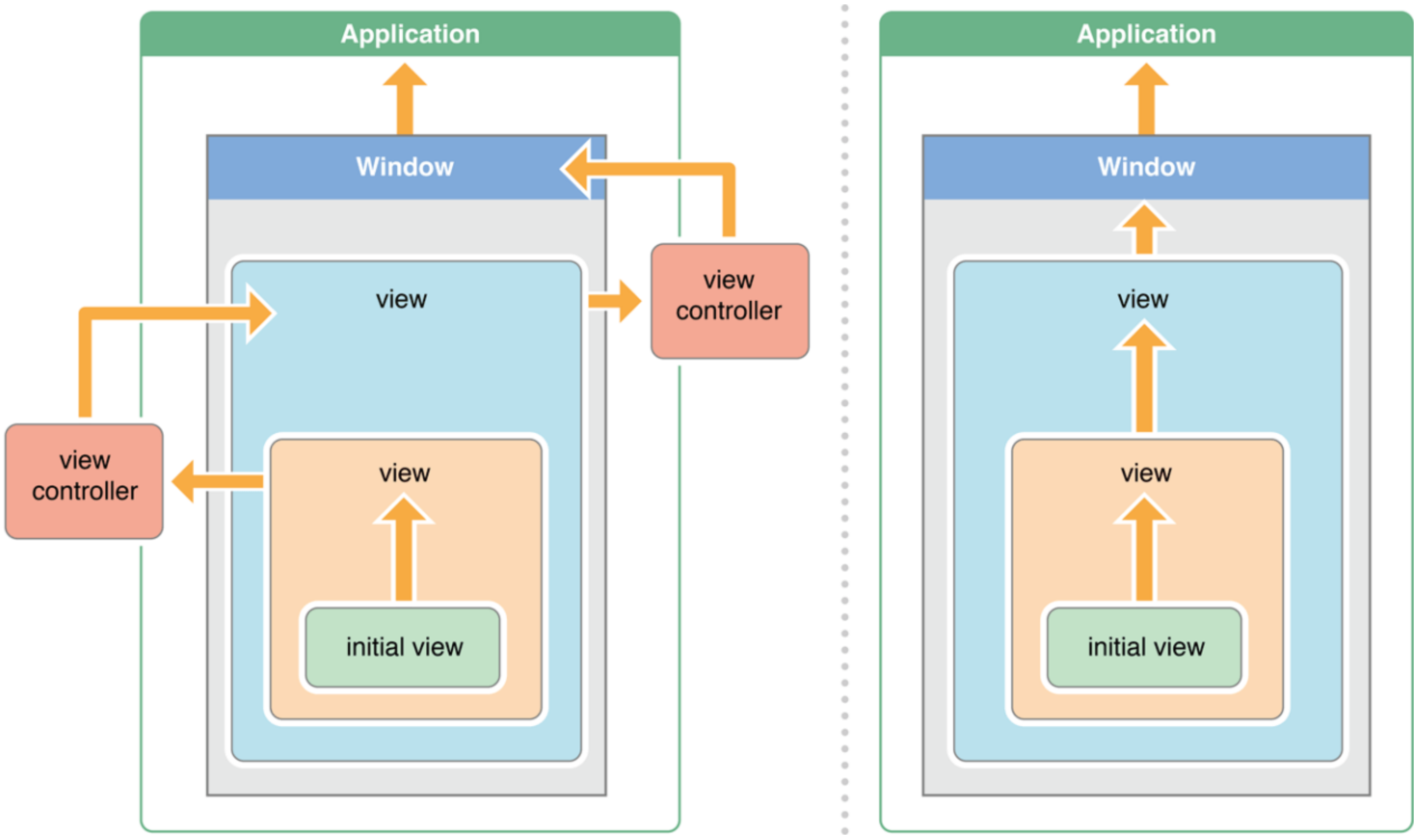
```
dispatch_queue_t myQueue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,
0);
dispatch_async(myQueue, ^{
    printf("Do some work here.\n");
});
```

Question 14

What is the Responder Chain?

When an event happens in a view, for example a touch event, the view will fire the event to a chain of `UIResponder` objects associated with the `UIView`. The first `UIResponder` is the `UIView` itself, if it does not handle the event then it continues up the chain to until `UIResponder` handles the event. The chain will include `UIViewController`s, parent `UIView`s and their associated `UIViewController`s, if none of those handle the event then the `UIWindow` is asked if it can handle it and finally if that doesn't handle the event then the `UIApplicationDelegate` is asked.

If you get the opportunity to draw this one out, it's worth doing to impress the interviewer:



Question 15

What's the difference between using a *delegate* and *notification*?

Both are used for sending values and messages to interested parties. A *delegate* is for one-to-one communication and is a pattern promoted by Apple. In *delegation* the class raising events will have a property for the *delegate* and will typically expect it to implement some `protocol`. The *delegating* class can then call the *delegate*s protocol methods.

Notification allows a class to broadcast events across the entire application to any interested parties. The broadcasting class doesn't need to know anything about the listeners for this event, therefore *notification* is very useful in helping to decouple components in an application.

```
[NSNotificationCenter defaultCenter]
          postNotificationName:@"TestNotification"
          object:self];
```

Question 16

What's your preference when writing UI's? Xib files, Storyboards or programmatic `UIView`?

There's no right or wrong answer to this, but it's great way of seeing if you understand the benefits and challenges with each approach. Here's the common answers I hear:

- Storyboard's and Xib's are great for quickly producing UI's that match a design spec. They are also really easy for product managers to visually see how far along a screen is.

- Storyboard's are also great at representing a flow through an application and allowing a high-level visualization of an entire application.
- Storyboard's drawbacks are that in a team environment they are difficult to work on collaboratively because they're a single file and merge's become difficult to manage.
- Storyboards and Xib files can also suffer from duplication and become difficult to update. For example if all button's need to look identical and suddenly need a color change, then it can be a long/difficult process to do this across storyboards and xibs.
- Programmatically constructing `UIView`'s can be verbose and tedious, but it can allow for greater control and also easier separation and sharing of code. They can also be more easily unit tested.

Most developers will propose a combination of all 3 where it makes sense to share code, then re-usable `UIViews` or `Xib` files.

Question 17

How would you securely store private user data offline on a device? What other security best practices should be taken?

Again there is no right answer to this, but it's a great way to see how much a person has dug into iOS security. If you're interviewing with a bank I'd almost definitely expect someone to know something about it, but all companies need to take security seriously, so here's the ideal list of topics I'd expect to hear in an answer:

- If the data is extremely sensitive then it should never be stored offline on the device because all devices are crackable.
- The keychain is one option for storing data securely. However it's encryption is based on the pin code of the device. User's are not forced to set a pin, so in some situations the data may not even be encrypted. In addition the users pin code may be easily hacked.
- A better solution is to use something like SQLCipher which is a fully encrypted SQLite database. The encryption key can be enforced by the application and separate from the user's pin code.

Other security best practices are:

- Only communicate with remote servers over SSL/HTTPS.
- If possible implement certificate pinning in the application to prevent man-in-the-middle attacks on public WiFi.
- Clear sensitive data out of memory by overwriting it.
- Ensure all validation of data being submitted is also run on the server side.

Question 18

What is MVC? How is it implemented in iOS? What are some pitfalls you've experienced with it? Are there any alternatives to MVC?

MVC stands for *Model, View, Controller*. It is a design pattern that defines how to separate out logic when implementing user interfaces. In iOS, Apple provides `UIView` as a base class for all *View*s, `UIViewController` is provided to support the *Controller* which can listen to events in a *View* and update the *View* when data changes. The *Model* represents data in an application and can be implemented using any `NSObject`, including data collections like `NSArray` and `NSDictionary`.

Some of the pitfalls that people hit are bloated `UIViewController` and not separating out code into classes beyond the MVC format. I'd highly recommend reading up on some solutions to this:

- https://www.objc.io/issues/1-view-controllers/lighter-view-controllers/
- https://speakerdeck.com/trianglecocoa/unburdened-viewcontrollers-by-jay-thrash
- https://programmers.stackexchange.com/questions/177668/how-to-avoid-big-and-clumsy-uitableviewcontroller-on-ios

In terms of alternatives, this is pretty open ended. The most common alternative is MVVM using ReactiveCocoa, but others include VIPER and using Functional Reactive code.

Question 19

A product manager in your company reports that the application is crashing. What do you do?

This is a great question in any programming language and is really designed to see how you problem solve. You're not given much information, but some interviews will slip you more details of the issue as you go along. Start simple:

- get the exact steps to reproduce it.
- find out the device, iOS version.
- do they have the latest version?
- get device logs if possible.

Once you can reproduce it or have more information then start using tooling. Let's say it crashes because of a memory leak, I'd expect to see someone suggest using *Instruments* leak tool. A really impressive candidate would start talking about writing a unit test that reproduces the issue and debugging through it.

Other variations of this question include slow UI or the application freezing. Again the idea is to see how you problem solve, what tools do you know about that would help and do you know how to use them correctly.

Question 20

What is AutoLayout? What does it mean when a constraint is "broken" by iOS?

AutoLayout is way of laying out `UIView`s using a set of constraints that specify the location and size based relative to other views or based on explicit values. *AutoLayout* makes it easier to design screens that resize and layout out their components better based on the size and orientation of a screen. *Constraint*s include:

- setting the horizontal/vertical distance between 2 views
- setting the height/width to be a ratio relative to a different view
- a width/height/spacing can be an explicit static value

Sometimes constraints conflict with each other. For example imagine a `UIView` which has 2 height constraints: one says make the `UIView` 200px high, and the second says make the height twice the height of a button. If the iOS runtime can not satisfy both of these constraints then it has to pick only one. The other is then reported as being "broken" by iOS.

Standard Questions

“Tell me about yourself”.

This is a REALLY tricky one! So many people read their resume out loud (forgetting I am looking right at it!), listing every class they took in college, every position they have ever held (and that really adds up for the Senior Level Engineers) and every responsibility (which often times happen to be the same across 2 or 3 companies that particular person has worked at. So, 20 minutes later, I still don't know anything new/outside of the resume about the candidate, we are 20 minutes in, and by then I believe that they might as well be robots (and robots tend to not fit very well with dynamic teams).

So do yourself (and the hiring manager) a favor by preparing a 2min long elevator pitch of your background- keep it simple, concise and brief, and then add something personal at the end. Do you love running? Are you coaching your daughter's soccer team? Die hard Star Wars fan? Adding a little bit of personality in is a great ice breaker, and will make for an amazing transition into the deeper part of the interview.

"Describe an interesting problem and how you solved it."

Your chance of hearing this one is basically 99%, so you better have a great answer ready for them! Here are a few tips: think of a particularly amazing project that you are proud of. Got it? Great! Now, grab a piece of paper and summarize it in 5 bullet points. 1st bullet point- very brief backstory of the circumstances behind the project 2nd bullet- the nature of the project itself 3rd bullet- the problems you have came across while solving it/them 4thbullet- how did you solve the difficulties above 5th bullet point- what have you learned from it.

Now, find somebody who will listen (a friend, partner, colleague) and ask them if the story makes sense (minus any technical aspects of it if the person you are talking with isn't your peer). It's a great way to validate if the story holds itself together.

You would be shocked how many times I have heard things like "and then I went to John, and he used the tool he has created to remove the critical bug which then……". Who is Tom? Your manager? Intern? Peer? Why did you go to him of everybody in your company? What is that magical tool that he has put together? Which critical bug exactly did it remove and how? What was the outcome?

You get the idea :)

Additional bonus? You now have 5 bullet points you can use as a cheat sheet during your interview without looking like THAT guy who is reading his resume out loud.

Now let's jump into the WEIRD questions that some hiring managers like to throw in to tip you off your game.

- How would you test a toaster?
- How many pens can you fit into an airplane?
- How many windows are in San Francisco?
- How many golf balls can fit in a school bus?
- How many Big Macs does McDonald's sell each year in the U.S.?

The list goes on and on. While it has nothing to do with your technical skills, it does a good job of checking how you deal with curve balls, and to see if you are capable of logical step-by step problem solving. In most cases, there aren't perfect answers to those type of questions (though you can google them for your peace of mind). It is all about how you explain your logic to your interviewer.

Here is the answer to the golf balls question by Michael Beauchamp, so you get the idea of what I am talking about:

> *I figure a standard school bus is about 8ft wide by 6ft high by 20 feet long - this is just a guess based on the thousands of hours I have been trapped behind school buses while traffic in all directions is stopped.*
>
> *That means 960 cubic feet and since there are 1728 cubic inches in a cubit foot, that means about 1.6 million cubic inches.*
>
> *I calculate the volume of a golf ball to be about 2.5 cubic inches (4/3 * pi * .85) as .85 inches is the radius of a golf ball.*
>
> *Divide that 2.5 cubic inches into 1.6 million and you come up with 660,000 golf balls. However, since there are seats and crap in there taking up space and also since the spherical shape of a golf ball means there will be considerable empty space between them when stacked, I'll round down to 500,000 golf balls.*

See? Step by step. And suddenly it's a relatively simple, logical question. Just stay calm, and instead of trying to come up with an instant number in your head, walk the interviewer through your train of thoughts with you.

Need real life practice? Feel free to schedule a mock interview with Matt, who has been doing technical interviews for 12 years, and ask for his feedback!