Possible Questions

Questions I can expect:

Q: Is architecture really important?
A: It depends. Many apps are disposable, throw-away apps, so if you don't expect to do many updates after v1.0, then it may be in your best interest to code it as quickly as possible and don't worry about the architecture. But for apps that reach a certain size, not doing any architecture may slow you down or even prevent you from finishing it. So it's a trade-off. No customer is going to praise you for your architecture or how beautiful the source code looks.

Q: Is architecture different between Swift and Objective-C?
A: Not really. Both are primarily OOP languages and all the Cocoa frameworks are OOP too. Swift lets you do functional programming, but that's possible too with Objective-C (but maybe a bit more cumbersome). The advantage of Swift is that it is more expressive, so you can make the architecture decisions more obvious, for example using enums with associated values. I'll be giving a talk on that exact topic immediately after this one, so check that out if you want to learn more about that.

Q: Modules?
A: It's useful to put the entire domain model into its own module. That requires you to think about what is part of the interface and what you keep hidden. Likewise, you can put the networking and persistence layers in their own modules.

Q: If you have many small classes, and many relationships between them, doesn't that make the architecture harder to understand rather than easier?
A: If the parts are small, then each part is easy to understand and test in isolation. But the big picture may become muddy indeed. The solution to this is to hide the small parts by creating abstractions such as layers, using design patterns such as Façade, etc.

Q: What is the difference between architecture and design patterns?
A: The architecture can use design patterns. It is the application of design patterns in a structured manner.

Q: Functional programming?
A: Swift and the Cocoa frameworks are primarily OOP but you can certainly use functional programming principles. Your domain model is something you can ask questions of, and functions are great for that. A function is basically a question. "Given this state, what would happen if I were to do so-and-so?"

The big thing about FP is immutability. We haven't really touched on that in this talk, but making your objects immutable is a great way to simplify your design, at the cost of runtime efficiency. You could make (parts of) your data model immutable, or the view-model. So every time something changes, you replace the view model with a completely new instance.

Q: What about things like Mantle, JSONModel, etc?
A: I think it's wise to make a distinction between the domain model and how those model objects are obtained and persisted. They are more than just containers of data.

Mantle creates objects based on JSON, but it's still a very thin model. I think it's useful for doing the JSON conversion to actual objects, but I consider this part of the data storage layer, not the domain layer. So, like with Core Data, you'd create your own wrapper to turn these objects into domain objects, and vice versa.

For many apps, a thin model is good enough, by the way.

Q: I don't like passing objects around all the time.
A: I believe it's better to make dependencies visible. It should be obvious what other objects a class depends on. Explicit is clearer than implicit in this case. It also makes it obvious when a class has too many dependencies.

Q: What if you use a third-party service like Parse (or CloudKit)?
A: You use their API and framework, and then convert these objects to your domain objects. It's no different from using Core Data to store objects. You provide adapters for the networking and data store, to hide them from your model layer. Of course, you could use NSManagedObject or PFObjects directly in your code but there are benefits to keeping those separate. (You could also make your data model objects extend from your NSManagedObject subclass etc.)

Q: Shouldn't the watchlist really be stored on the server?
A: Probably, yes. It depends on the app, I suppose. But you may not have the choice. If the server is controlled by someone else, and they don't offer this feature in their API, you'll have to store it locally (or perhaps on your own server). The point I'm trying to make here is that apps are not just dumb clients and that all the interesting domain stuff happens on the server. In many cases, the app will have its own, slightly different view of the world.

Q: Why do Bid and Item need a == operator (and conform to Equatable)?
A: This is needed for when you search again. These are different Item objects but may have the same ID as your existing Item objects in your watchlist. (They may potentially have different bids that your existing ones, if you haven't pulled in the latest bids. I won't solve this syncing issue in this talk.)

Q: What happened to Bidder?
A: There is no object for Bidder at the moment; the app doesn't really do anything with individual bidders yet, so I left it out of the domain model for now.

Q: Why not singletons?
A: Global state. Creates a dependency at compile time. That dependency is hard to see because it's kind-of hidden in the code.

Q: How to do authentication/login screen?
A: Authentication is done by ServerAPI. If you try to send a request but are not authenticated yet, or when the server says you're not authenticated anymore, the ServerAPI will notify the app (through a delegate method), which will then pop-up a login screen. The ServerAPI holds on to the original request, and if the login completes, it sends that request. (That's why you want to create these Request objects, so that you can hold on to them and queue them etc.)

Q: PeriodicCheckForBids and ListOfItems are strange names for classes.
A: You might be tempted to call this "LatestBidsPoller" or something instead. Both names describe what the class does. In classic OOP, a class is supposed to be a thing, and its name should be the name of that thing. Like ServerRequest is a thing. But sometimes classes are better described by what they do. What do you do? I poll the latest bids. If I could come up with the name of an object that checks for something every so often, I'd use that, but "poller" isn't it.