iOS Dev Accelerator
Week 1 Day 1

- Intro to Course
- MVC
- JSON
- Bundles
- TableView/Cells

Brad

Andy

Leo

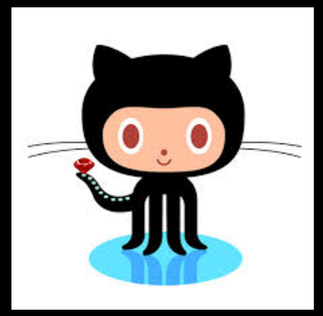
Course Format

- From 9 to noon, we are downstairs in the east room for lecture time.
- From noon to 4, we are upstairs working on the homework.
- Each week we create a separate standalone app. All the lectures and homework are based around each app's features.
- There is new homework every day, but you will submit your homework only once at the end of each week, by Sunday morning at 8am.
- You will be a given a score based on how many of the required features you got implemented, if your app builds, are you following best practices, etc

Weeks 1–4 Outline

Week 1 - Twitter Clone

Week 2 - Photo Filtering



Week 3 - Github Client

Week 4 - First Project Week

Weeks 5–8 Outline

Week 5 - Location Aware Reminders

Week 6 - Objective-C StackOverflow Client

Week 7 - Swiftify (Spotify clone)

Week 8 - 2nd Project Week w/ Javascripters

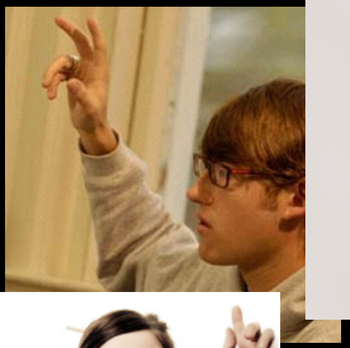
Themed Days

- Mobile Mondays: Overview of the Android version of what we learned in the prior week
- Teach others Tuesday: Blog about the things you have learned!
- Web-tools Wednesday: Learn how to write your own backend with nodeJS and javascript
- Technical Thursdays: Learn the data structures and algorithms necessary for interviews
- Friendship Fridays: Pair up and tackle tough extra challenges on the homework app

Passing this class with your job guarantee

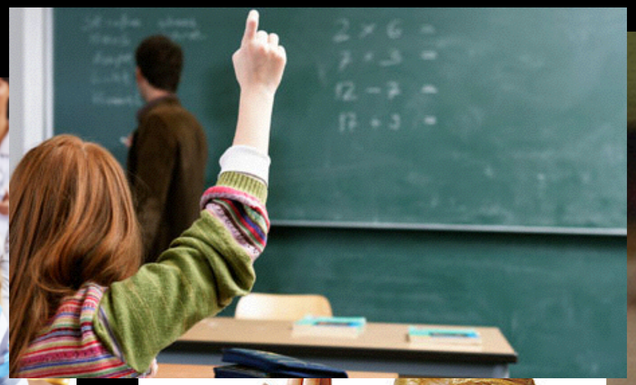
- 3 things you must do to receive your your job guarantee:
    1. Complete all homework assignments with a final average of over 90%
    2. Pass a final whiteboard exam (you get more than one try)
    3. At least one app submitted to the app store, but hopefully two

Tips for doing well





Ask all the questions

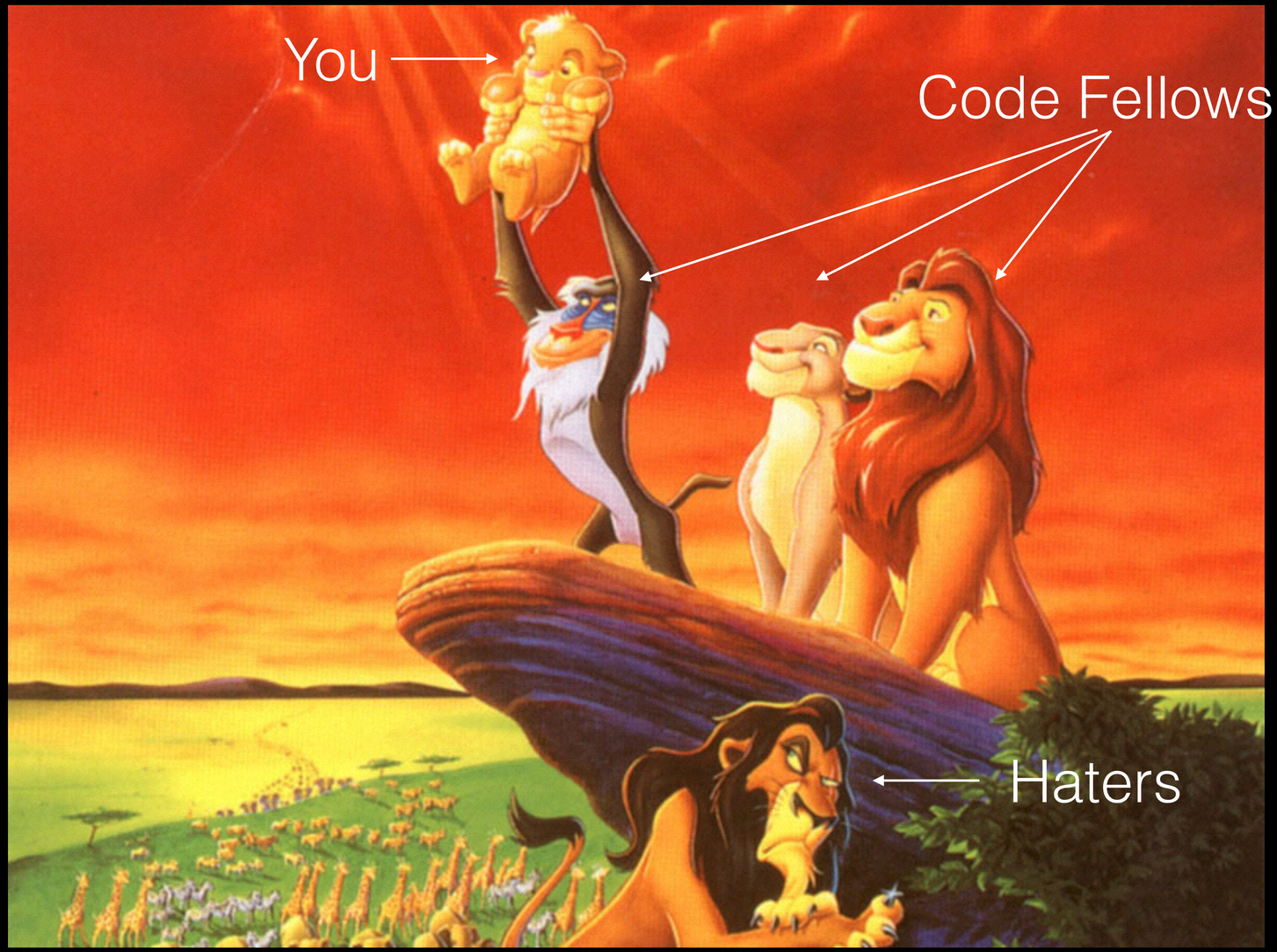

Solid Note Taking

- All slides, code, and recordings are posted immediately after lectures are over.
- So instead of blindly typing everything we type or everything on the slides, try to just absorb the information into your brain.
- Scientific Studies have shown students typically have the best results if they take notes on a notepad during class and keep their laptops closed.
- Science is never wrong so you should probably close your laptop.

Don’t get stuck

- Remember to always consult fellow students and your teachers if you are stuck on something or if something is unclear.
- You can always look at the sample code from the lectures as well.
- Knowing how to Google your coding problems is a very important skill as a developer, so always try Googling as well. Chances are someone has asked that exact same question on stack overflow or another forum.

Help us Help you

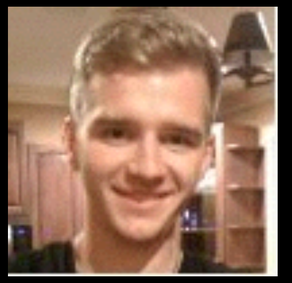


Tim Hise, Nordstrom

Michael Babiy, Getty Images

Chris Meehan, HCL Technologies

Rich Lichkus, Reveal



Ivan Lesko, General UI

Lauren Lee, Urban Spoon

Anton Rivera, Big Fish Games

Andrea Silkey, Disney

Brian Radebaugh, Nordstrom

Chris Cohan, Eigital

Steven Stevenson, Belief

Dan Fairbanks, Fresh Consulting

Zuri Biringer, Siren

Andrew Rodgers, L4 Mobile

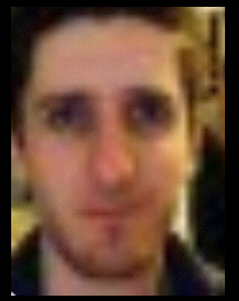
Reed Sweeney, LIFFFT

Matt Remick, Nordstrom

Spencer Fornaciari, Blank Check Labs

Ryo Tulman, Muegello

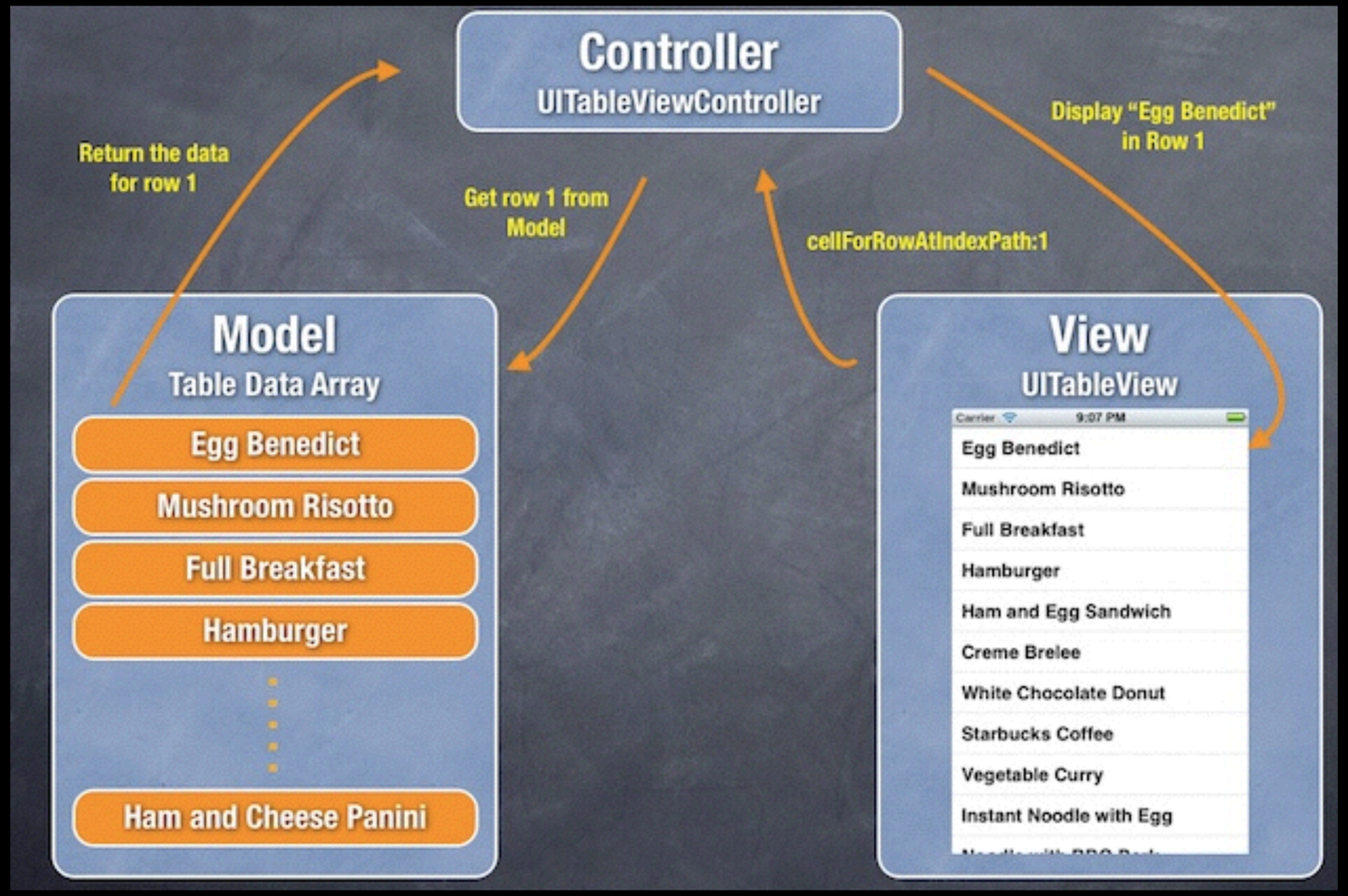
Christian Hansen, National Center of Telehealth



Jeff Schwab, Felt

lectures

https://gitter.im/bradleypj823/CFiOSChat



MVC
(Model-View-Controller)

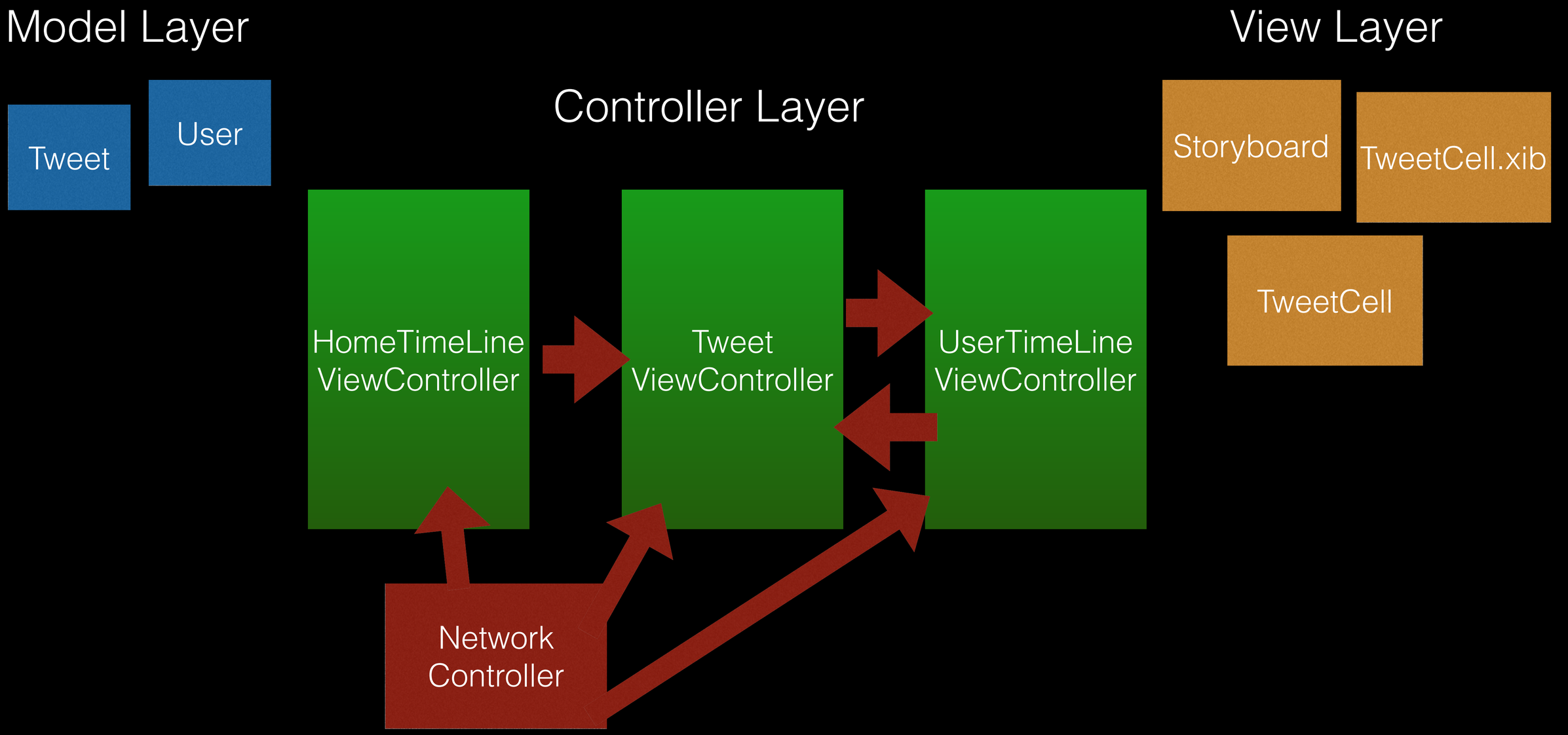
MVC Facts

- Introduced in the 70's with the Smalltalk programming language.
- Didn't become a popular concept until the late 80's
- The MVC pattern has spawned many evolutions of itself, like MVVM (Model-View-ViewModel)
- MVC is very popular with web design and applications. It's not just for mobile or desktop.

So what is MVC?

- MVC is simply the separation of **M**odel, **V**iew, and **C**ontroller.
- It is a separation of concerns for your code. Being able to separate out these components makes your code easier to read, re-use, test, think about, and discuss.
- The **Model layer** is the data of your app, the **View layer** is anything the user sees and interacts with, and the **Controller layer** mediates between the two.
- Examples of bad MVC practices: Your model classes directly communicating with your views, your view classes making network calls, etc

The MVC layout of our Week 1 App



Demo

JSON

JSON

- “JavaScript Object Notation” (but it really is language independent)
- “open standard format that uses human readable text to transmit data objects consisting of attribute-value pairs”
- Used primarily for communication between the server and client
- Is a more popular alternative to XML
- The official internet media type of JSON is application/json (remember for later on in this course)

JSON data types

- **Number** : Decimal number that makes no distinction between an integer and float
- **String** : A sequence of zero or more unicode characters. Delimited with double quotation marks, and escaped with a backslash
- **Boolean** : True or false vales
- **Array**: An ordered list of zero or more values, can be of any type. Array's use [] square bracket notation with elements separated by a comma.
- **Object** aka **dictionary** : Unordered associative collection. Use {} curly bracket notation with pairs separated by a comma. Within each pair separation is established with a : colon. All keys must be strings and must be unique within that object.
- **null** : empty value.

JSON Example

- An example of a JSON response From StackOverflows API
- The root object is an dictionary denoted by the curly bracket {
- The first pairing is the key “items” with a corresponding array with just one item, a dictionary.
- For this api, each question is represented by a dictionary in this array. I only asked for one.


```
{
  "items": [
    {
      "tags": [
        "ios",
        "objective-c",
        "uiwebview",
        "uiscrollview",
        "screen-orientation"
      ],
      "owner": {
        "reputation": 22,
        "user_id": 3751662,
        "user_type": "registered",
        "accept_rate": 40,
        "profile_image": "https://www.gravatar.com/avatar/fb058f5e726691830e2aca67d0201a55?
        "display_name": "Daljeet",
        "link": "http://stackoverflow.com/users/3751662/daljeet"
      },
      "is_answered": false,
      "view_count": 8,
      "answer_count": 1,
      "score": 0,
      "last_activity_date": 1406309661,
      "creation_date": 1406308178,
      "question_id": 24960983,
      "link": "http://stackoverflow.com/questions/24960983/uiwebview-content-is-lost-partia
      "title": "UIWebView content is lost partially on changing orientation?"
    }
  ],
  "has_more": true,
  "quota_max": 10000,
  "quota_remaining": 9995
```

JSON workflow in your app

1. Make a network call and receive raw JSON data back.
2. Serialize the JSON data into foundation objects (dictionaries, arrays, strings,etc)
3. Parse through the serialized JSON objects and create your model objects

JSON Parsing – interpreting JSON in your code

- You may eventually use third party frameworks to simplify your JSON parsing, but first you need to understand how to write your own parsing.
- It's conceptually similar to parsing through a plist.
- Use the NSJSONSerialization class to convert raw JSON data you get back from a network service to Foundation objects and vice versa.

NSJSONSerialization

`+ JSONObjectWithData:options:error:`

Returns a Foundation object from given JSON data.

Declaration

SWIFT

```swift
class func JSONObjectWithData(_ data: NSData!,
                              options opt: NSJSONReadingOptions,
                                error error: NSErrorPointer) -> AnyObject!
```

OBJECTIVE-C

```objc
+ (id)JSONObjectWithData:(NSData *)data
                 options:(NSJSONReadingOptions)opt
                   error:(NSError **)error
```

Parameters

data	A data object containing JSON data.
opt	Options for reading the JSON data and creating the Foundation objects. For possible values, see NSJSONReadingOptions.
error	If an error occurs, upon return contains an `NSError` object that describes the problem.

NSJSONSerializationOptions

- NSJONReadingMutableContainers : specifies that arrays and dictionaries are created as mutable objects
- NSJSONReadingMutableLeaves : Specifies that leaf strings in the JSON Object graph are created as instances of NSMutableString
- NSJSONReadingAllowFragments : Specifies that the parses should allow top level objects that are not instance of NSArray or NSDictionary
- NSJSONWritingPrettyPrinted: Specifies that the JSON data generated should use white space to make it more human readable.

JSON and Optionals

- Optionals play a big part in JSON handling in Swift.
- You are going to use *Optional Binding and Down-casting* a lot when parsing through JSON.
- Optional Binding can be used to find out if an optional contains a value, and if it does then it makes that value available as a temporary constant.

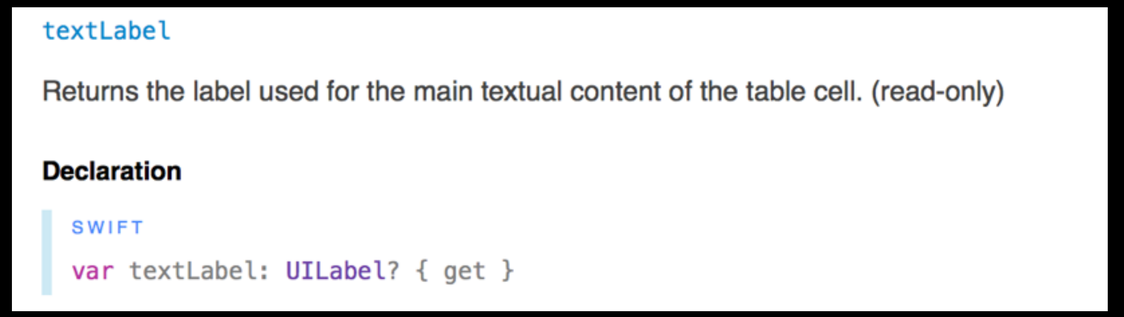
Optionals Review

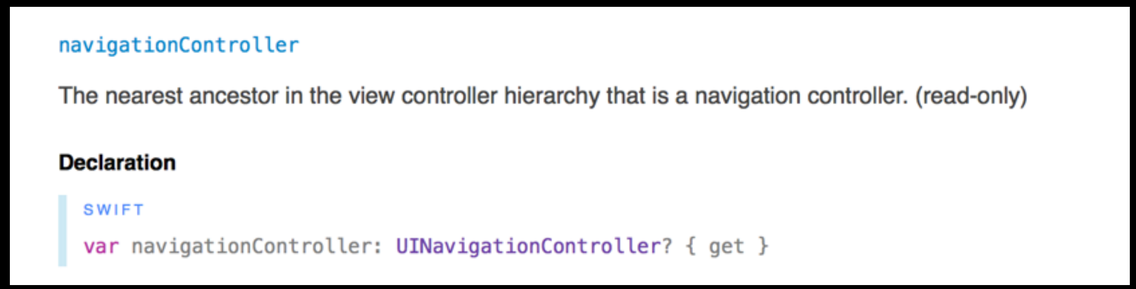
- You use optionals in situations where a value may be absent. An optional says:
  - There is a value, and it equals x
  - or
  - There isn't a value at all
- A variable is marked as optional by adding a question mark to the end of its type:

```
var twitterAccount : ACAccount?
```

Optionals Review

- You will see a ton of optionals in Apple's API's, specifically for return values or properties:





- So in your code, you will need to account for the fact that these values can be nil. So how do we do that?

Checking optionals for values

- you can use an if statement to check if an optional contains a value:

```
if convertedNumber != nil {
    println("convertedNumber contains some integer
        value.")
}
```

- Once you know an optional contains a value, you can use the ! to access its underlying value. This is called a **forced unwrapping** :

```
if convertedNumber != nil {
    println("convertedNumber has an integer value
        of \(convertedNumber!).")
}
// prints "convertedNumber has an integer value of
        123."
```

Optional Binding

- You can use **optional binding** to find out wether an optional contains a value,and if so, to make that value available as a temporary constant or variable that is unwrapped.
- The syntax of an optional binding:

```
if let constantName = someOptional {
    statements
}
```

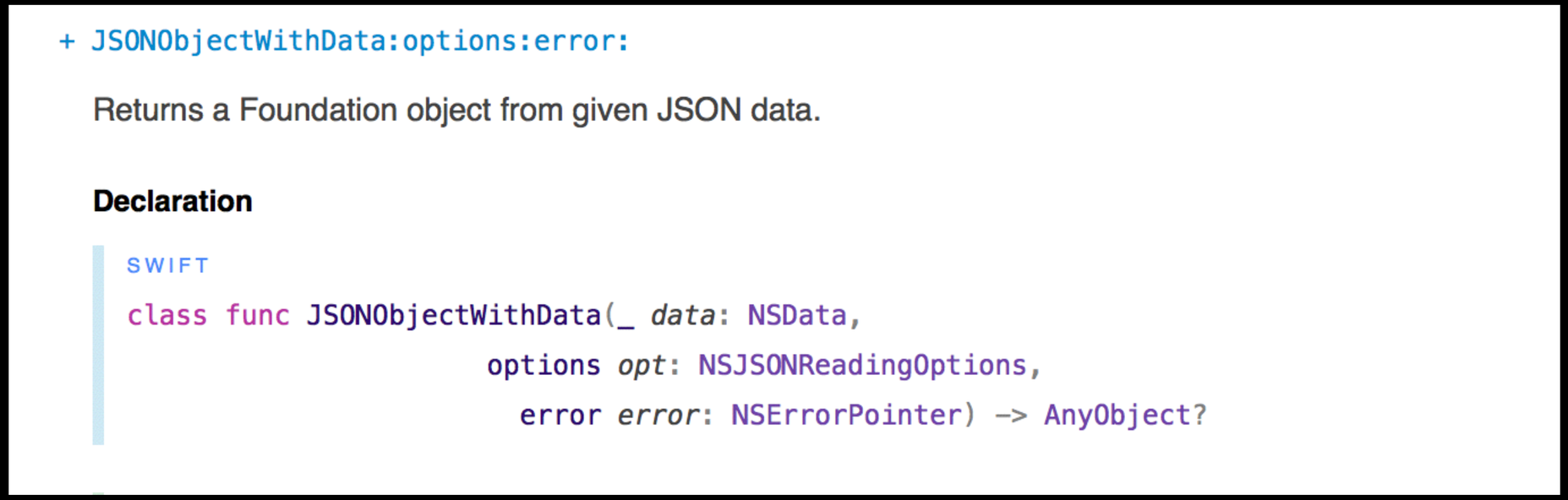
using optional binding

```
func printTitleValue(value : String?) {

    if let title = value {
        println(title)
    }
}
```

not using optional binding

```
func printTitleValue(value : String?) {

    if value != nil {
        let title = value!
        println(title)
    }
}
```

JSON and Optionals



- As you can see here, the NSJSONSerialization method that we use to convert the raw JSON data to foundation objects returns an optional value.
- So we first must check to see if a value was even returned.
- And then we check to see if the value returned matches the type of object we are expecting.

Downcasting

- So we can combine optional binding and downcasting to achieve those checks in one line.
- Downcasting is used whenever a constant or variable of a certain type may actually refer to an instance of a subclass behind the scenes.
- When you think this is the case, you can use downcasting to attempt to cast the variable or constant to the subclass.
- There are two forms of down casting:
  - optional form: as?
  - forced form : as

JSON and Optionals

```
let JSONData = NSData(contentsOfFile: path!)

if let JSONArray = NSJSONSerialization.JSONObjectWithData
    (JSONData, options: nil, error: &error) as? NSArray {
    //converting the raw jsonData to an Array worked, we can
        now parse through it
} else {
    //converting to array didnt work, oh no :(
}
```

Demo

Bundles

Bundles

- “Bundle are a fundamental technology in OS X and iOS that are used to encapsulate code and resources”
- A bundle is a directory with a standard hierarchical structure that holds code and resource for code.
- Bundles provide programming interfaces for accessing the contents of bundles in your code.

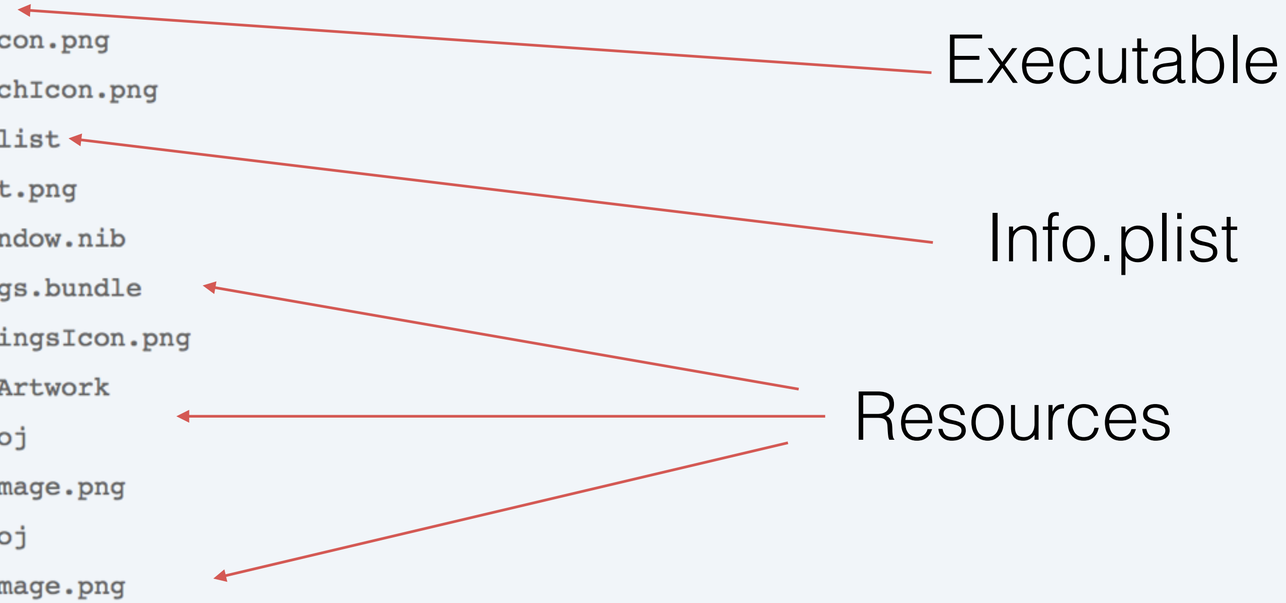
Application Bundle

- There a number of different types of bundles, but for iOS apps the most important is the application bundle.
- The application bundle stores everything your app requires to run successfully.
- Inside of your application bundle lives 4 distinct types of files:
  - Info.plist - a plist file that contains configuration information for your application. The system relies on this to know what your app is.
  - Executable - All apps must have an executable file. This file has the apps main entry point and any code that was statically linked to your app’s target.
  - Resource files - Any data that lives outside your app’s executable file. Images,icons, sounds, nibs, etc. Can be localized.
  - Other support files - Mostly used for mac apps. Plugins, private frameworks, document templates.

Application Bundle

Listing 2-1 Bundle structure of an iOS application

```
MyApp.app
    MyApp
    MyAppIcon.png
    MySearchIcon.png
    Info.plist
    Default.png
    MainWindow.nib
    Settings.bundle
    MySettingsIcon.png
    iTunesArtwork
    en.lproj
        MyImage.png
    fr.lproj
        MyImage.png
```



Bundles in code

- the NSBundle class represents a bundle in code.
- NSBundle has a class method called mainBundle(), which returns the bundle that contains the code and resources for the running app.
- You can also access other bundles that aren't the main bundle by using bundleWithPath() and passing in a path to another bundle.
- You can get the path for a resource by using pathForResource(ofType:)

UITableView and UITableViewCell

TableViews

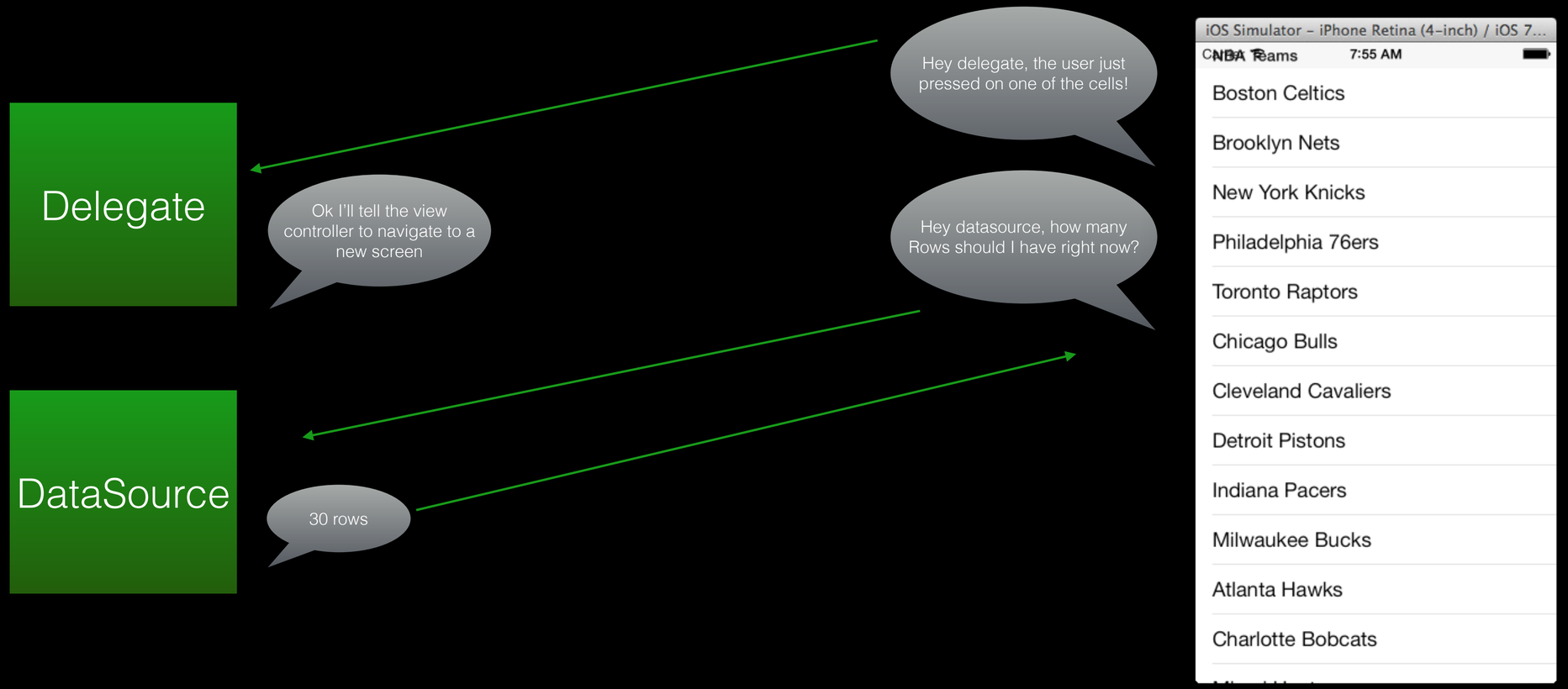
- “A Tableview presents data in a scrollable list of multiple rows that may be divided into sections.”
- A Tableview only has one column and only scrolls vertically.
- A Tableview has 0 through n-1 sections, and those sections have 0 through n-1 rows. A lot of the time you will just have 1 section and its corresponding rows.
- Sections are displayed with headers and footers, and rows are displayed with Tableview Cells, both of which are just a subclass of UIView.

So how do Tableviews work?

- Tableviews rely on the concept of delegation to get their job done.

- Picture time:

TableViews and Delegates



A tableview has 2 delegate objects. One actually called delegate and one called datasource. The datasource pattern is just a specialized form of delegation that is for data retrieval only.

Delegation

- The whole point of delegation is to allow you to implement custom behavior without having to subclass.
- Apple could have designed UITableView's API so you would have to subclass UITableView, but then you would have to understand UITableviews in a lot more detail(which methods can I override? Do I have to call super?)
- Delegates adopt a protocol to declare that they are capable of being the delegate. Think of it like a contract.
- Delegation is used extensively in a large portion of Apple's frameworks.

TableViews

- A tableview requires 2 questions to be answered (aka methods to be implemented) by its delegates and they are all in the datasource.
- `tableView(numberOfRowsInSection:) How many rows am I going to display?`
- `tableView(cellForRowAtIndexPath:) What cell do you want for the row at this index?`
- Number of sections is actually optional, and is 1 by default.

Demo

TableViewCell

- UITableViewCell is a direct subclass of UIView.
- You can think of it as a regular view that contains a number of other views used to display information.
- The ‘Content View’ of a cell is the view that all content of a table view cell should be placed on. Think of it as the default super view of your cell. It contentView itself is read only.

TableViewCell Style

- Setting the style of an instance of UITableViewCell will expose certain interface objects on the cell.
- The default style exposes the default text label and optional image view.
- Right Detail exposes a right aligned detail text label on the right side of the cell in addition to the default text label.
- Left Detail exposes a left aligned detail text label on the right side of the cell in addition to the default text label.
- Subtitle exposes a left aligned label below the default text label.

Creating tableView Cells

- You can instantiate them in code with the initializer init(style: UITableViewCellStyle, reuseIdentifier: String?)
- But usually you will be setting them up in your storyboard or in a xib file.
- If they are in your storyboard, you just have to set their reuse identifier in the identity inspector, and then call dequeueReusableCellWithIdentifier() at the appropriate time.
- Later in the week we will learn how to design and use cells with xibs.

Demo